# MAPSI

Modèles et Algorithmes de Probabilités et Statistiques pour l'Informatique



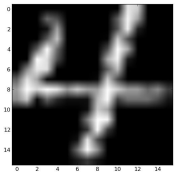
# Apprentissage de paramètres par max de vraisemblance

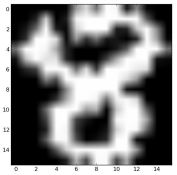
Dans ce TME, l'objectif est d'apprendre grâce à l'estimateur de maximum de vraisemblance les paramètres de lois normales à partir d'un ensemble de données. Ces lois normales seront ensuite exploitées pour faire de la classification (comme nous l'avions vu en cours avec les images de désert, forêt, mer et paysages enneigés).

Ici, notre base de données d'apprentissage est la base **USPS**. Celle-ci contient les images réelles de chiffres provenant de codes postaux écrits manuellement et scannés par le service des postes américain. Ces données scannées ont été normalisées de manière à ce qu'elles soient toutes des images de 16x16 pixels en teintes de gris, cf. Le Cun et al., 1990:

Y. LeCun, O. Matan, B. Boser, J. S. Denker, *et al.* (1990) "Handwritten zip code recognition with multilayer networks". In ICPR, volume II, pages 35–40.

Voici quelques exemples d'images de cette base :





## 1. Préparation / visualisation

Vous allez tout d'abord télécharger la base qui va vous servir pour votre apprentissage de paramètres : **USPS training**. Le format de ce fichier est décrit dans **format** mais, pour simplifier le TME, nous vous proposons ci-dessous une fonction qui permet de lire le fichier usps. Cette fonction renvoie un tableau de tableaux d'images. Chaque image est un tableau numpy de 256 nombres réels compris entre 0 et 2 qui caractérisent l'intensité des pixels dans l'image (16x16 = 256 pixels). Les images correspondant au même chiffre manuscript sont placées dans un même tableau. Le fichier usps indique en effet pour chaque image à quel chiffre celle-ci correspond. Dans le jargon de l'apprentissage, les chiffres sont appelés **classes** et, lorsque nous aurons de nouvelles images dont nous essayerons de déterminer, grâce à nos lois normales, à quel chiffre elles correspondent, nous dirons que nous faisons de la **classification**. Enfin, apprendre les paramètres de nos lois normales à partir d'un fichier qui contient pour chaque image sa classe s'appelle de l'**apprentissage supervisé**. Pour terminer la description de l'objet retourné par la fonction **read_file**, les tableaux d'images correspondant à chaque chiffre sont eux-mêmes stockés dans un tableau et c'est ce dernier qui est renvoyé. Chaque élément de ce tableau correspond donc à l'ensemble des images d'une classe. Le premier élément contient ainsi toutes les images du chiffre 0, le 2ème toutes celles du chiffre 1, et ainsi de suite. Par exemple, si **read_file**("fichier") retourne un objet training_data, alors training_data[2] est le tableau de toutes les images du chiffre 2, training_data[2][3] correspond à la 4ème image du chiffre 2, autrement dit à un tableau de 256 nombres réels représentant cette image.

```
# -*- coding: utf-8 -*-

import numpy as np

def read_file ( filename ):
    """
    Lit un fichier USPS et renvoie un tableau de tableaux d'images.
    Chaque image est un tableau de nombres réels.
    Chaque tableau d'images contient des images de la même classe.
    Ainsi, T = read_file ( "fichier" ) est tel que T[0] est le tableau
    des images de la classe 0, T[1] contient celui des images de la classe 1,
    et ainsi de suite.
    """
    # lecture de l'en-tête
    infile = open ( filename, "r" )
    nb_classes, nb_features = [ int( x ) for x in infile.readline().split() ]

    # creation de la structure de données pour sauver les images :
    # c'est un tableau de listes (1 par classe)
    data = np.empty ( 10, dtype=object )
    filler = np.frompyfunc(lambda x: list(), 1, 1)
    filler( data, data )

    # lecture des images du fichier et tri, classe par classe
    for ligne in infile:
        champs = ligne.split ()
        if len ( champs ) == nb_features + 1:
            classe = int ( champs.pop ( 0 ) )
            data[classe].append ( map ( lambda x: float(x), champs ) )
    infile.close ()

    # transformation des list en array
    output  = np.empty ( 10, dtype=object )
    filler2 = np.frompyfunc(lambda x: np.asarray (x), 1, 1)
    filler2 ( data, output )

    return output
```

[$[Get Code]]

Afin de visualiser les images que vous allez manipuler, nous vous proposons ci-dessous une fonction **display_image** qui prend en argument une image (c'est-à-dire un tableau de 256 nombres réels) et qui l'affiche dans une fenêtre. Exécutez cette fonction sur quelques images de votre base d'apprentissage afin de visualiser les données que vous allez manipuler par la suite.

```
import matplotlib.pyplot as plt

def display_image ( X ):
    """
    Etant donné un tableau de 256 flotants représentant une image de 16x16
```

SEARCH
Go

TUTORIEL NUMPY

INFOS COURS/TD/TME

SEMAINIER

Semaine 1
Semaine 2
Semaine 3
Semaine 4
Semaine 5
Rapport 1
Semaine 6
Semaine 7?
Semaine 8
Semaine 9?
Semaine 10?

LIENS

edit SideBar

```
    pixels, la fonction affiche cette image dans une fenêtre.
    """
    # on teste que le tableau contient bien 256 valeurs
    if X.size != 256:
        raise ValueError ( "Les images doivent être de 16x16 pixels" )

    # on crée une image pour imshow: chaque pixel est un tableau à 3 valeurs
    # (1 pour chaque canal R,G,B). Ces valeurs sont entre 0 et 1
    Y = X / X.max ()
    img = np.zeros ( ( Y.size, 3 ) )
    for i in range ( 3 ):
        img[:,i] = X

    # on indique que toutes les images sont de 16x16 pixels
    img.shape = (16,16,3)

    # affichage de l'image
    plt.imshow( img )
    plt.show ()
```

[$[Get Code]]

## 2. Maximum de vraisemblance pour une classe

Dans ce TME, nous allons étudier la distribution de probabilité des teintes de gris des images (en fait, la fonction de densité car on travaille sur des variables aléatoires continues) . Nous allons faire l'hypothèse (certes un peu forte mais tellement pratique) que, dans chaque classe, les **teintes des pixels sont mutuellement indépendantes**. Autrement dit, si $X_i$ , $i$ =0,...,255, représente la variable aléatoire "intensité de gris du *i*ème pixel", alors $p(X_0, \ldots, X_{255})$ représente la fonction de densité des teintes de gris des images de la classe et:

$$p(X_0, \ldots, X_{255}) = \prod_{i=0}^{255} p(X_i).$$

Ainsi, en choisissant au hasard une image dans l'ensemble de toutes les images possibles de la classe, si celle-ci correspond au tableau `np.array([x_0,...,x_255])`, où les `x_i` sont des nombres réels compris entre 0 et 2, alors la valeur de la fonction de densité de l'image est égale à $p$(`x_0,...,x_255`) = $\prod_{i=0}^{255} p$(`x_i`).

Nous allons de plus supposer que chaque $X_i$ suit une distribution normale de paramètres $(\mu_i, \sigma_i^2)$ (autrement dit, $p$(`x_i`) = $\mathcal{N}(\mu_i, \sigma_i^2)$). Par maximum de vraisemblance, estimez, pour une classe donnée, l'ensemble des paramètres $(\mu_0, \ldots, \mu_{255})$ et $(\sigma_0^2, \ldots, \sigma_{255}^2)$. Pour cela, écrivez une fonction **learnML_class_parameters** ( classe ) qui, étant donné le tableau d'images d'une classe tel que retourné par la fonction **read_file**, renvoie un couple de tableaux, le premier élément du couple correspondant à l'ensemble des $\mu_i$ et le 2ème à l'ensemble des $\sigma_i^2$, $i$ =0,...,255. C'est-à-dire que **learnML_class_parameters** ( classe ) renverra un objet similaire à :

( array ( $[\mu_0, \ldots, \mu_{255}]$ ), array ( $[\sigma_0^2, \ldots, \sigma_{255}^2]$ ) )

## 3. Maximum de vraisemblance pour toutes les classes

En utilisant la fonction de la question précédente, écrivez une fonction **learnML_all_parameters** ( train_data ) qui, étant donné le tableau retourné par la fonction **read_file** (donc contenant toutes les images de toutes les classes), renvoie une **liste** de couples ( array ( $[\mu_0, \ldots, \mu_{255}]$ ), array ( $[\sigma_0^2, \ldots, \sigma_{255}^2]$ ) ). Vous exécuterez cette fonction sur vos données d'apprentissage et sauvegarderez le résultat dans une variable **parameters**.

## 4. Vraisemblance d'une image

Nous allons maintenant tester si, étant donné de nouvelles images, on peut classer celles-ci correctement, c'est-à-dire si on peut retrouver les chiffres auxquelles elles correspondent. Pour cela, nous allons utiliser de nouvelles images se trouvant dans le fichier **USPS test**. Ce fichier a exactement le même format que celui d'apprentissage et peut donc être lu grâce à la fonction **read_file**. En particulier, pour chaque image, nous avons le chiffre auquel elle correspond, ce qui nous permettra de vérifier que notre classifieur fonctionne correctement. Téléchargez le fichier et lisez-le en utilisant **read_file**.

Ecrivez une fonction **log_likelihoods** ( image, parameters ) qui, étant donné une image (donc un tableau de 256 nombres réels) et l'ensemble de paramètres déterminés dans la question précédente, renvoie un tableau contenant, pour chaque chiffre possible, la log-vraisemblance qu'aurait l'image si celle-ci correspondait à ce chiffre. Ainsi, si tab = **log_likelihoods** ( image, parameters ), tab est un tableau de 10 éléments (les 10 chiffres possibles) et tab[3] est égal à la log-vraisemblance de l'image dans la classe "chiffre = 3". **Attention:** pour certains pixels de certaines classes, la valeur de $\sigma^2$ est égale à 0 (toutes les images de la base d'apprentissage avaient exactement la même valeur sur ce pixel). Dans ce cas, la vraisemblance de toute image sur ce pixel doit être de 1 (et donc sa log-vraisemblance doit être égale à 0).

## 5. Classification d'une image

Ecrivez une fonction **classify_image ( image, parameters )** qui, étant donné une image et l'ensemble de paramètres déterminés dans la question 3, renvoie la classe la plus probable de l'image, c'est-à-dire celle dont la log-vraisemblance est la plus grande.

## 6. Classification de toutes les images

Ecrivez maintenant une fonction **classify_all_images** ( test_data, parameters ) qui, étant donné l'ensemble des images du fichiers USPS test tel que retourné par la fonction **read_file** et l'ensemble de paramètres déterminés dans la question 3, renvoie un tableau numpy bi-dimensionnel **T** de taille 10x10 tel que **T**[i,j] représente le pourcentage d'images correspondant dans la réalité au chiffre i que votre classifieur a classées dans la classe j (pour tout i, $\sum_{j=0}^{9}$ **T**[i,$j$] = 100%).

## 7. Affichage du résultat des classifications

Afin de visualiser les résultats obtenus par votre classifieur, exécutez la fonction suivante, qui prend en paramètres le tableau obtenu à la question précédente. Si votre classifieur est performant, vous devriez observer des pics sur la diagonale.

```
from mpl_toolkits.mplot3d import Axes3D

def dessine ( classified_matrix ):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x = y = linspace ( 0, 9, 10 )
    X, Y = np.meshgrid(x, y)
```

```
    ax.plot_surface(X, Y, classified_matrix, rstride = 1, cstride=1 )
```

[$[Get Code]]

Page last modified on October 03, 2014, at 09:30 AM EST